DOI: http://dx.doi.org/10.7551/978-0-262-32621-6-ch042

Evolving Morphologies with CPPN-NEAT and a Dynamic Substrate

Daniel Richards and Martyn Amos

School of Computing, Mathematics & Digital Technology, Manchester Metropolitan University
D.Richards@mmu.ac.uk

Abstract

Recent advances in fabrication technologies open up exciting opportunities to manufacture entirely new types of physical materials and structures. These have varied and specific mechanical properties, which can be exploited in a number of engineering applications, and a growing area of research concerns the generation of two- and three-dimensional designs using these materials. However, the *computational tools* required to explore large spaces of possible 2-D and 3-D morphologies remain underdeveloped. State-of-the-art evolutionary approaches such as CPPN-NEAT and HyperNEAT-LEO are often used to explore possible 2-D and 3-D designs, but their ability to construct efficient solutions for *practical* use in engineering domains remains in question. In this paper, we present an extension of CPPN-NEAT, in which nodes grow connections across a *dynamic substrate*, and illustrate this by creating efficient 2-D truss structures. Using four benchmark problems, we then demonstrate that our extended CPPN-NEAT model outperforms similar HyperNEAT methods for approximating specific connectivity patterns, and suggests important clues regarding how to best harness generative and developmental representations to build scalable and high-performance physical morphologies.

Introduction

Processes of evolution and development have shaped a vast array of physical structures. The desire to construct *artificial* structures with similar levels of complexity and efficiency is a driving force behind much of contemporary engineering. Recent advances in manufacturing technologies, specifically additive manufacturing (i.e. 3-D printing), open up the very real possibility of fabricating high-performance, physical designs that exhibit complex bio-inspired morphologies and behaviors [1-3]. Critically, this enhanced ability to control *how* and *where* material is distributed within structures enables the construction of entirely new classes of materials and objects, for use in various engineering domains. For example, at small scales, bone tissue scaffolds can be fabricated with bioactive glass to exhibit specific mechanical properties, such as high resistance to fracture [4]. At larger scales, these advances suggest vast potential for applications such as flexible cellular microstructures for prosthetic limbs [5], morphing wing designs for aerospace applications [6], and next-generation architectural designs [7,8].

While new fabrication hardware is facilitating exciting opportunities for engineering domains, the computational tools needed to fully exploit these technologies and aid discovery of functional morphologies with novel mechanical properties, remain relatively underdeveloped.

As noted by Hiller and Lipson [9], well-known structural optimization algorithms, such as homogenization techniques [10], can already successfully address simple design problems such as 2-D and 3-D truss structures. However, because they rely on prior knowledge of how to exploit local gradient information, they are limited in their ability to discover complex designs that meet higher-level *functional* goals.

To address this limitation, *evolutionary algorithms* are often used to explore large design spaces and generate efficient solutions. Evolutionary methods are potentially useful because they can explore search spaces when no *problem specific knowledge* exists. However, evolutionary algorithms have their own set of limitations when applied to engineering domains. These include: lack of scalability [11], limited ability to ensure buildable solutions [12], inability to guarantee global optima [13] and difficulty in applying them to design *exploration* (i.e. beyond late-stage parameter optimization) [14].

Various work based on the NEAT (Neuroevolution of Augmenting Topologies) model (specifically, CPPN-NEAT [15, 24-26] and HyperNEAT [27, 28]) suggests the existence of exciting opportunities for addressing many of these issues, critically offering key advantages in terms of scalability. However, there still exist significant challenges relating to the use of these approaches for building 2-D and 3-D morphologies for engineering domains. For example, Devert et al [20] have recently highlighted critical "approximation accuracy" limitations of HyperNEAT when applied to 2-D truss structure optimization. Additionally, Fenton el al [32] note that ANN-based methods are less desirable than grammatical systems, due to the difficulties of imposing *physical constraints*.

In this paper, we precisely address these two major issues and present an approach that combines NEAT with Compositional Pattern-Producing Networks (CPPN). Specifically, this approach exploits *growth* and *relative targeting* of connections on a *dynamic substrate* to: (A) create 2-D truss structures that out-perform those produced by similar HyperNEAT approaches on four well-known topology optimization benchmark problems, and (B) enable simple ways of embedding problem-specific constraints for engineering applications. We first review related work, and then present our model, demonstrating how it extends a typical CPPN-NEAT method. We describe our experimental setup and present our results, before concluding with a discussion and suggestions for further work.

Background

Generative and developmental representations offer a powerful framework for creating 2-D and 3-D designs [16,17]. Kicinger et al [18], present a generative representation, based on cellular automata, to evolve tall steel structures for building design. Kowaliw et al [19] use a novel embryogeny to evolve 2-D truss structures. Recently, Devert et al [20] demonstrate a novel ontogenic approach for 2-D truss optimization. For a comprehensive review of evolutionary structural design methods, see [21] and [22].

Recent work, particularly within the area of evolutionary robotics, demonstrates the construction of diverse 2-D and 3-D morphologies by evolving CPPNs [15] with Stanley and Miikkulainen's NEAT algorithm [23]. For example, Clune and Lipson [24] use CPPN-NEAT to evolve 3D printed objects; Cheney et al [25], evolve virtual creatures with novel morphologies comprising multiple materials; Hiller and Lipson [9], evolve multi-material physical objects to meet high-level functional goals, such as specific deformations of 3-D beams; Auerbach and Bongard [26] evolve virtual creatures with diverse locomotive behaviors; and Szerlip and Stanley [27] use HyperNEAT-LEO [28,29] to evolve diverse, functional 2-D designs for the simulation engine *Sodarace*.

We suggest that NEAT-based approaches offer significant potential for use in engineering domains, by enabling the discovery of novel material compositions with *specific mechanical properties* and *high-level functionality*. However, as highlighted by Devert et al [20], HyperNEAT currently has problems with accurately generating specific connectivity patterns, and this limitation makes it difficult to optimize even relatively simple 2-D truss structures. At the core of this issue is that HyperNEAT struggles to produce *modular* networks. Verbancsics and Stanley [29] show that by adding a bias towards creating local connections, HyperNEAT-LEO can create modular solutions and improve performance on various control problems. However, we believe that this approach, in its current form, is not enough to create efficient 2-D and 3-D morphologies – we support this claim in the Results section.

The second key challenge for CPPN-NEAT, when applied to engineering problems, is the inclusion of *physical constraints*. For example, to apply standard structural analysis techniques, solutions must have all parts suitably connected (i.e. no disconnected elements), yet dealing with fully connected solutions can be computationally expensive. For this reason, grammatical systems are often the favored generative approach to evolving buildable 2-D and 3-D solutions. Notably, Hornby and Pollack [30] evolve L-System grammars to create buildable table designs; Rieffel and Smith [31] use a grammatical approach to grow soft-bodied robot morphologies; and Fenton et al [32] use grammatical evolution to optimize simple truss structures and restrict the design space to a set of standard member sizes.

The key insight in this paper is that CPPN-NEAT can control simple (grammar-like) growth rules that play out on a 2-D (or 3-D) grid and generate local connectivity patterns, which are interpreted as physical designs. The benefit of this extra layer of abstraction is that we can: (A) more accurately approximate specific connectivity patterns; (B) easily enforce physical constraints and (C) exploit CPPN-NEAT's scalability and capacity to create geometric regularities.

Methods

Representation

CPPN-NEAT and HyperNEAT have already been described in detail [15, 24-29, 33], so we provide only a brief summary, and mainly focus on how our proposed method differs from previous work. CPPNs are similar to neural networks, yet nodes may contain a variety of different mathematical functions and can be evolved using NEAT [23]. CPPN-NEAT can be used to evolve complex geometric patterns. For example, 2-D patterns can be drawn by querying a CPPN and setting the color of each pixel on a canvas as a function of its x and y coordinates. Similarly, HyperNEAT uses CPPNs to specify the weight, w, of all connections in a larger fully connected feed-forward neural network (termed the "substrate", to distinguish it from the CPPN, which is also a network), by querying the coordinates of each input node i and output node j. That is, for each connection, the CPPN inputs x_i, y_i, x_j, y_j and outputs w.

The key idea in this paper is to use CPPNs to create 2-D morphologies in a similar manner to HyperNEAT (as [27]), but with *one key difference*. That is, instead of querying a fully connected *fixed* substrate of possible connections and specifying the dimensions of individual truss members, we use CPPNs to generate growth instructions for each node as a function of its x and y coordinates (Fig 1).

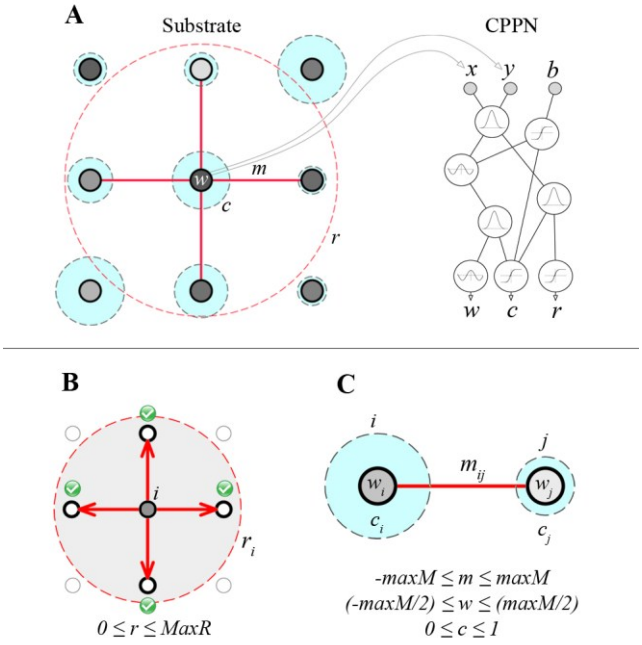

Figure 1. Growth of structural elements. (A) Nodes are initialized with growth instructions by querying a CPPN. Nodes grow connections using three properties: *r*, *w*, *c*. (B) Each node has a range of influence, *r*, and is permitted make connections between all other nodes which are within this radius. (C) Nodes also have a weight, *w*, and a concentration, *c*. Connections between nodes *i* and *j* have a cross-sectional area, m_{ij}, which is created by summing w_i and w_j, with a potential bias that is determined by concentrations c_i and c_j.

The shift from directly querying connections (specific targeting) to querying nodes and then *growing* connections (relative targeting of connections) provides at least three interesting possibilities when generating network structures. Firstly, it imposes a hard-coded bias towards creating local connections. That is, unlike [29], in this approach the ability to create longer connections actively increases the number of possible connections in the entire structure, thus enlarging the dimensionality of the problem. This means that networks with shorter, local connections are often easier to optimize, and thus more likely to emerge. Secondly, *fixed* fully connected substrates can be replaced by *dynamic substrates,* where the maximum connectivity of any individual node varies across the substrate as a function of growth. Notably, the bottleneck in many engineering problems is the computation time required for evaluation, so eliminating the need to simulate fully connected substrates can result in significant savings of CPU time. Thirdly, as we have previously shown [34], growth provides a useful mechanism for imposing necessary physical constraints. For example, during growth, individual nodes can be limited to a maximum or minimum number of connections.

As shown in Figure 1, nodes are initialized by querying a CPPN (using Cartesian coordinates as inputs), and then grow connections on a larger substrate, where: $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$. The CPPN uses three inputs (x and y coordinates and one bias) and returns three output values (w, c, r). Each connection has a cross-sectional area, m, which is defined by summing the weight, w, from each endpoint node and potentially applying a small bias based on the concentration value, c, of both end nodes. Thus, to define the cross-sectional area m_{AB} we use:

$$
\begin{aligned}
&bias = (max(c_A,c_B) - min(c_A,c_B)) \times Q \\
&if(c_A \geq 0.5 \text{ and } c_B < 0.5) \\
&\quad m_{AB} = (w_A \times (1 + bias)) + (w_B \times (1 - bias)) \\
&else\ if(c_A < 0.5 \text{ and } c_B \geq 0.5) \\
&\quad m_{AB} = (w_A \times (1 - bias)) + (w_B \times (1 + bias)) \\
&else\ m_{AB} = w_A + w_B
\end{aligned}
\qquad (1)
$$

Where m_{AB} is the cross-sectional area of the connection between nodes A and B, c is the concentration value, w is the weight of each node, and Q in a fixed coefficient value used to determine the maximum bias. In this paper, we set Q=0.2 for all problems.

Each node has a minimum and maximum range, r. To avoid matrix errors during structural analysis, we enforce a minimum degree of connectivity for all nodes (Fig 2B).

There are two significant implications of this; firstly, nodes can be completely "switched off" by reducing the r below the *minR* threshold. This means that no other nodes can connect, and this provides a useful mechanism for sculpting network topology. Secondly, because nodes contain useful information about their immediate neighborhood, *null connections* that do not contribute to the node's minimum connectivity are easily eliminated, excluding them from computationally expensive analysis. For example, fully connected solutions (Fig 2C), where all members have null cross-sectional areas ($m \leq 1.e^{-7}$), can be converted into Figure 2B before evaluation, which is more computationally efficient to simulate. Note, if minimum connections are defined as "null", we use a very small cross-sectional area ($m = 1.e^{-7}$), as is standard in truss optimization.

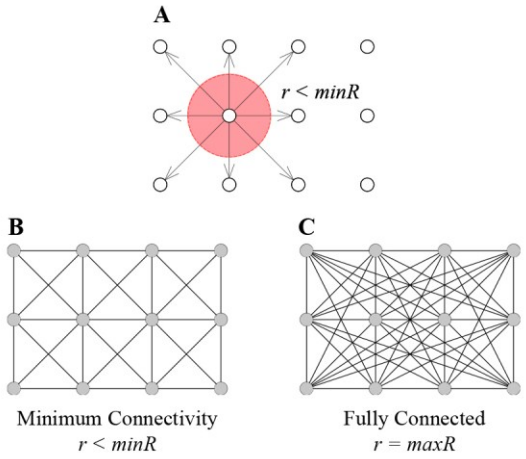

Figure 2. Enforcing minimum and maximum connectivity. (A) Each node must at least connect to all of its immediate neighbors. This distance is defined as the minimum connectivity limit: *minR*. If a node's $r < minR$, all connections made to this node become null and it is connected to its immediate neighbors. (B) Minimum connectivity of all nodes. (C) Fully connected structure.

Truss structures (as shown in Fig 2) must not contain overlapping (duplicate) connections; therefore to enable a dynamic substrate, further constraints must be applied to ensure that such connections are disallowed. To achieve this, we modify our "data-tag" approach used during node interactions, described in detail in [34]. As in [34], nodes swap and manage a small set of data-tags to keep track of their existing and new connections. Data-tags are checked before new connections are made, to avoid creating duplicates. Additionally, to avoid overlapping connections, we constrain nodes so that they are only permitted to connect to neighbors when there are no intermediate nodes.

Static Analysis

To evaluate the 2-D structures we use the linear *direct stiffness method* [35], which is a common finite element method (FEM). The process involves modeling the stiffness properties of each truss member and using the information to assemble a larger global stiffness matrix, $K(a)$, which describes the mechanical behavior of the entire structure:

$$K(a) = \sum_{j=1}^{n} a_j K_j$$

Where a_j is the cross-sectional area of the *jth* truss element and K_j is the member stiffness of the *jth* truss element. When M forces ($f_1, \ldots, f_M$) are imposed on nodes in the truss they react and move. The displacement, u, of loaded nodes is solved with the linear system:

$$K(a)u_k = f_{k,} \qquad k = 1, \ldots, M$$

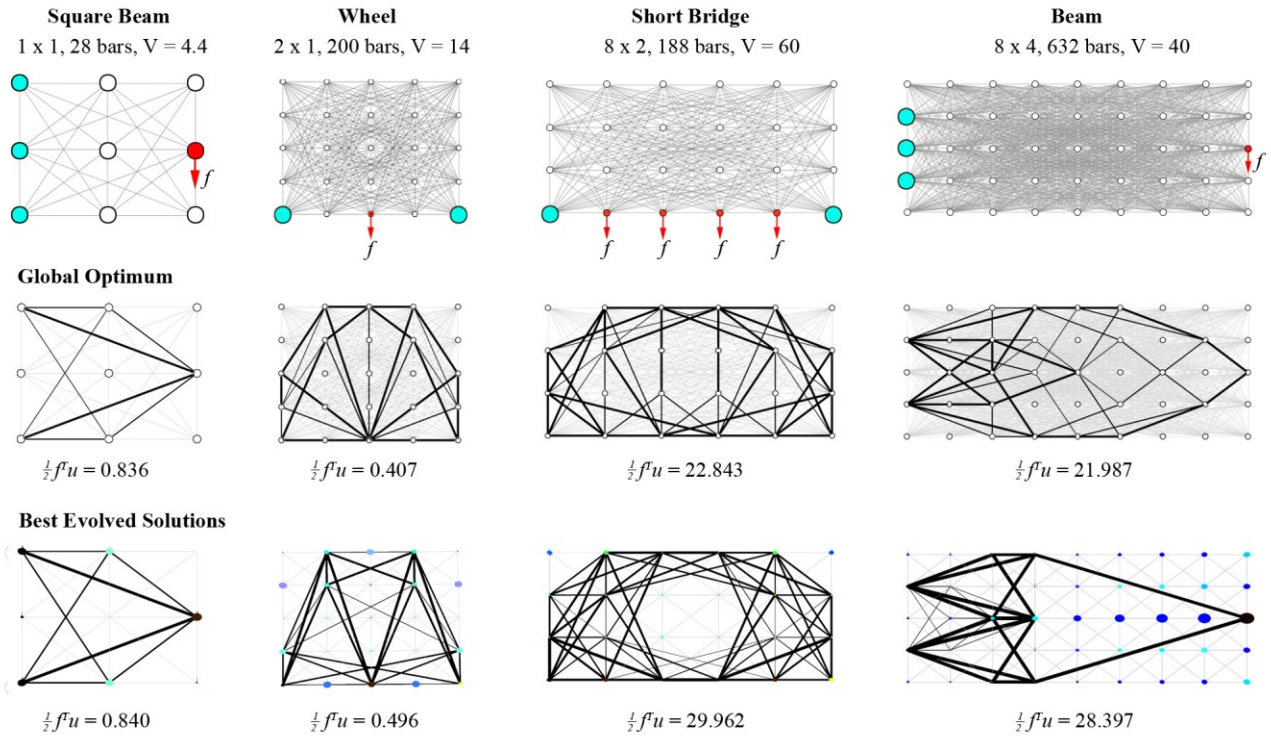

Figure 3. Benchmark Problems. (Top row) Four benchmark problem instances. Blue nodes have restricted degrees of freedom, and red nodes have an imposed one unit load in the direction of the arrow. The descriptions above give (a) *x* and *y* dimensions of truss (note: figures show the *substrate*, not the absolute *x* and *y* dimensions of each truss as used in FEM analysis), (b) number of bars and (c) the volume constraint, *V* (see [36] for further details). (Middle row) Global optima (from [36]). (Bottom row) Best evolved solutions using CPPN-NEAT with dynamic substrate. Here node color and size represent the node properties *c* and *r*, respectively.

Benchmark Problems

We test our method using the same truss benchmark problems as those used by Devert et al [20] (Fig 3), and compare our results with similar HyperNEAT methods.

The benchmark problems are relatively simple, have known global optima [36], and can be solved perfectly using evolutionary approaches with direct representations. However, as highlighted by Devert et al [20], HyperNEAT (*HyperNEAT 3.0 C++* package with default parameters) struggles to generate good solutions to these problems. They argue that, due to this limitation, ontogenic representations are superior. However, we suggest that such ontogenic representations are only applicable when problems have valuable *gradient information* that is already *known* to be exploitable. For example, Devert et al's ontogenic encoding uses 32 FEM calls per solution and works by iteratively querying a CPPN, each time inputting the structural strain of each truss member and returning an incremental modification of the cross-sectional area. This technique does indeed provide superior solutions to similar generative representations. However, when such gradient information is readily available and the correlation between parameters is well understood (i.e. if member strain is low, reduce cross-sectional area), existing homogenization techniques are generally orders of magnitude faster [10, 37].

Since our motivation is to explore large search spaces where useful gradient information is largely *unknown,* and homogenization techniques do not exist, we argue that better generative representations will provide a valuable way of discovering complex material compositions in various problem domains [1-9]. Consequently, we compare our method against: (1) Devert et al's recorded HyperNEAT results [20], (2) HyperNEAT with a locality seeded LEO [29], and (3) our model *without* the dynamic substrate, where *r* is binary either "on" with full connectivity, or "off" ($r < minR$) with all null connections (see Fig 2).

We use the common topology optimization objective:

$$\min \frac{1}{2} f^T u$$

Subject to the constraints:

$$\sum_{j=1}^{n} a_j l_j \leq V$$

$$a \in [0,1]$$

Where: a_j is the cross-sectional area (a continuous value between 0 and 1) of the *jth* bar, l_j is the length of the *jth* bar, and *V* is a problem specific volume constraint (see Fig 3).

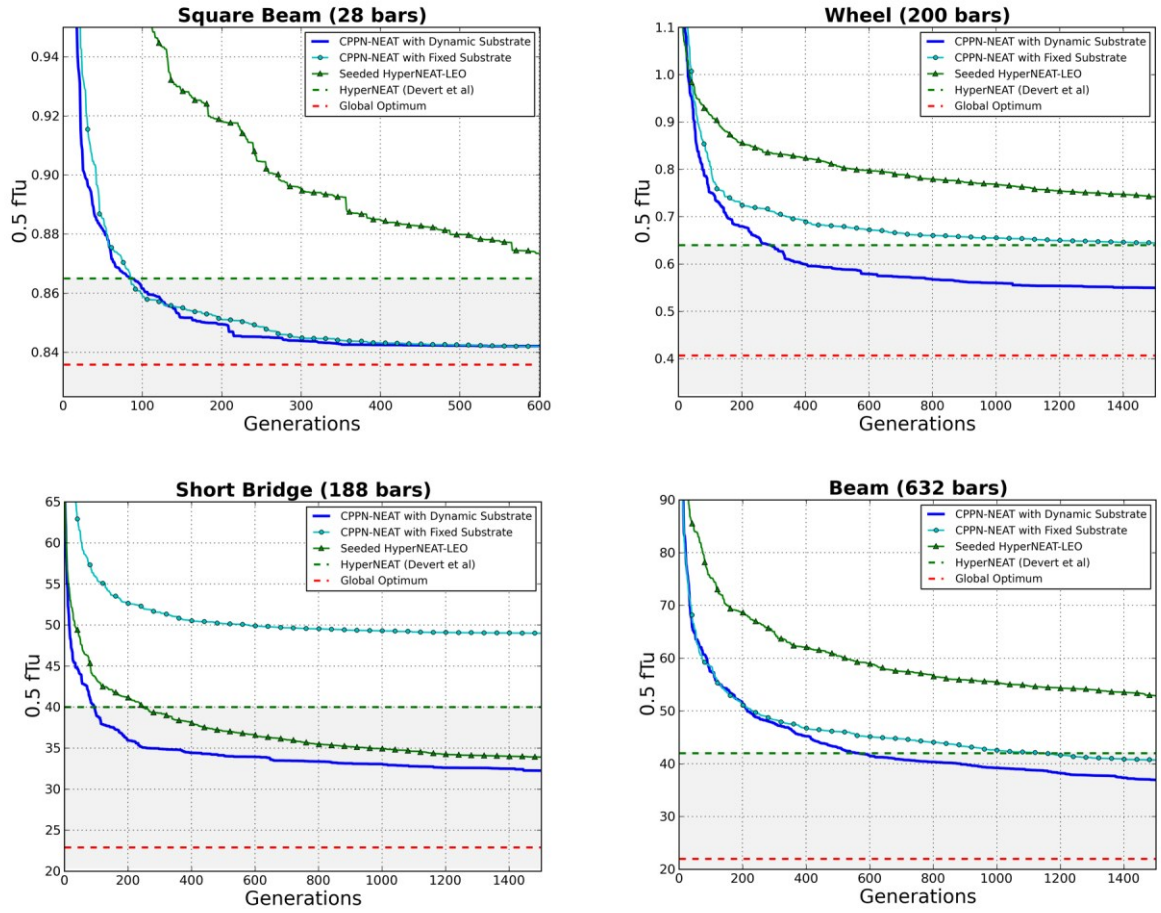

Figure 4. Median solutions obtained using CPPN-NEAT with dynamic substrate, compared against a typical HyperNEAT method (indiated as a fixed threshold value, showing the median best solutions obtained by [20] over an equivalent number of FEM evaluations, from 64 independent runs). HyperNEAT with a seeded LEO [29], our CPPN-NEAT with fixed substrate, and the known global optimum (taken from [36] – indicated as a dotted line). On all problems, CPPN-NEAT with a dynamic substrate performs best.

Experimental Details

For all benchmark problems, we perform 20 independent runs, each with a population of 150 CPPNs, evolved for a maximum of 1500 generations. As NEAT is a maximization algorithm, we adapt the previously outlined objective function to define our fitness function:

$$max \ \frac{1}{\frac{1}{2}f^T u}$$

We use our own java implementation of NEAT and HyperNEAT-LEO. CPPNs use the following activation functions: *Gaussian, Cosine and Sigmoid* with equal probability of being produced. We promote 25% of the population with elitism, and there is an 80% chance of mutating individuals after crossover.

Mutation rates are 0.03, 0.05 and 0.8 for adding a new node, adding a new link and perturbing connection weights, respectfully – and probability of interspecies mating is 0.001. We use a dynamic compatibility threshold, where the target number of species is 10 and the niche size required for elitism is 5. Finally, we set compatibility coefficients to: $c_1 = 2.0$, $c_2 = 2.0$, $c_3 = 1.0$.

The Young's modulus of all non-null beams is 1.0 and for null members is 0.0. All force loads (shown with a red arrow in Fig 3) have a magnitude of 1.0 and are in the direction of the arrow. Each problem has a maximum volume of material, *V*, if any solution has a *volume* > *V*, the cross-sectional area of all members are linearly scaled to meet *V*. The minimum cross-sectional area of any truss member is $1.0e^{-7}$ and anything below this value is null. Finally, for all of these experiments, the maximum cross-sectional area (*maxM*) is 1.0.

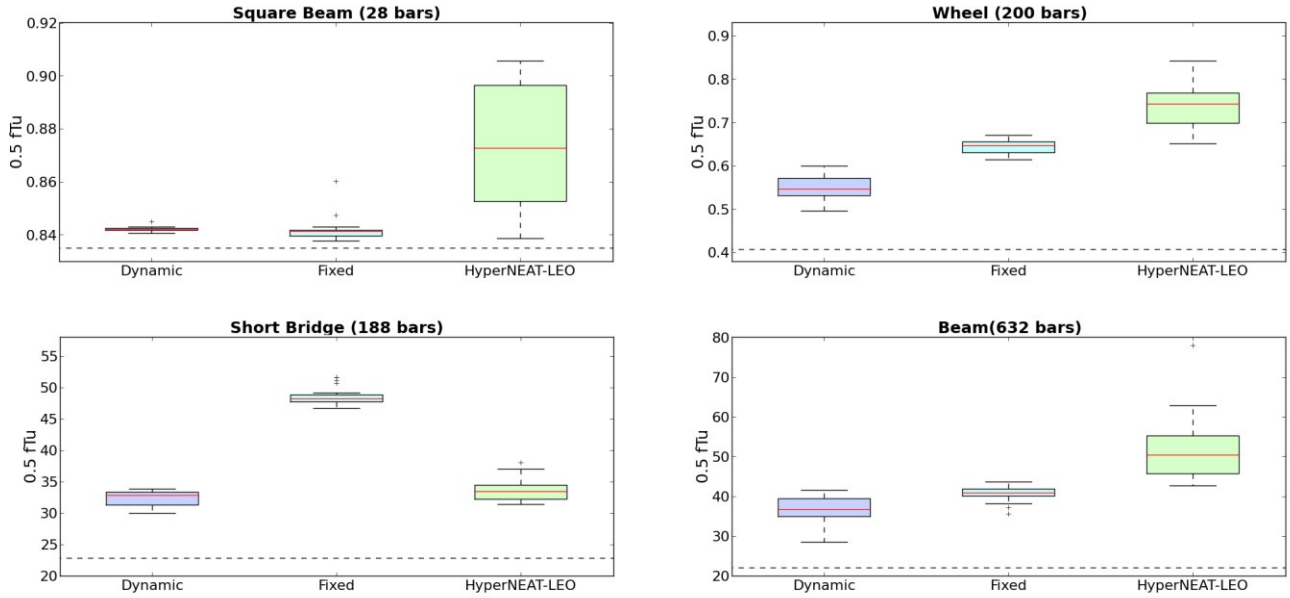

Figure 5. Best solutions obtained for different problems over 20 independent runs for CPPN-NEAT with dynamic substrate, CPPN-NEAT with fixed substrate and the seeded HyperNEAT-LEO. Dotted lines indicate global optima.

Results

Our results demonstrate that CPPN-NEAT with dynamic substrate outperforms both standard HyperNEAT and seeded HyperNEAT-LEO on all four benchmark problems. Specifically, the dynamic property of the substrate, coupled with the ability to silence nodes during growth (i.e. when $r <$ *minR,* see Fig 2), enables our CPPN-NEAT method to create better truss solutions and more closely approximate specific connectivity patterns of the known global optimum (Fig 3).

Figure 4 shows the median solutions obtained using our dynamic method in comparison to: (A) standard HyperNEAT (as described in [20]), (B) HyperNEAT-LEO with a seeded bias towards expressing local connections [29], and (C) our CPPN-NEAT method using a fixed and fully connected substrate. As shown, the CPPN-NEAT with dynamic substrate converges faster than all other methods and quickly improves on the best solutions found by [20]. Figure 5 shows the best solutions obtained with each approach; we see that CPPN-NEAT with dynamic substrate improves on the best median solutions generated with the seeded HyperNEAT-LEO by 3.5% on the smallest *square beam* problem, a more substantial 26.5% on the *wheel*, 2.8% on the *short-bridge* and 27% on the largest *beam* problem.

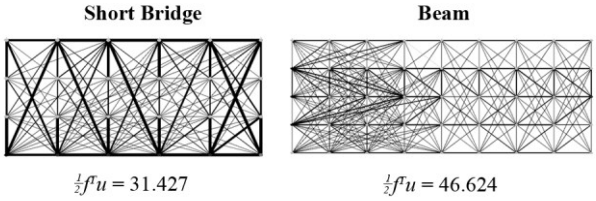

Figure 6. Best truss designs obtained with the seeded HyperNEAT-LEO over 20 independent runs.

Interestingly, the seeded HyperNEAT-LEO performs well on the *short-bridge* problem and significantly outperforms standard HyperNEAT and CPPN-NEAT with fixed substrate. We attribute this increased performance to the substrate configuration of the problem. That is, the short-bridge problem uses a 5 x 3 grid of nodes, yet the x and y dimensions of the substrate of the are always equal ($-1 \leq x \leq 1$ and $-1 \leq y \leq 1$). This means that, unlike the *square-beam* and *wheel* problem, nodes are not equally spaced in the substrate, so here the HyperNEAT-LEO with seeded bias for creating local connections is able to improve solutions. However, as shown in Figures 4 and 5, the seeded HyperNEAT-LEO actually performs much worse on the larger *beam* problem, which has similar (8 x 4) substrate configuration. To understand this difference, we can look at the type of truss designs produced by seeded HyperNEAT-LEO (Fig 6). Here we see that a key problem is that it struggles to eliminate non-essential connections, and instead tends favor highly connected designs. This tendency becomes increasingly detrimental to truss optimization as the problems scale up and we increase the number of possible connections.

Critically, CPPN-NEAT with dynamic substrate *does not* suffer from this problem, and, in contrast, evolves solutions with much more clearly differentiated network morphologies (Fig 3). Interestingly, we observe that during the early stages of evolution, CPPN-NEAT with dynamic substrate tends to create truss structures with minimal connectivity patterns that become more complex over time (similar to NEAT) as structurally significant nodes in the substrate become densely connected and establish longer connections.

These results suggest that while seeding HyperNEAT-LEO with a bias towards creating local connections is clearly a valuable method for improving performance and modularity in ANN problem domains [39], when creating 2-D (and most likely 3-D) designs, CPPN-NEAT with dynamic substrate provides superior solutions.

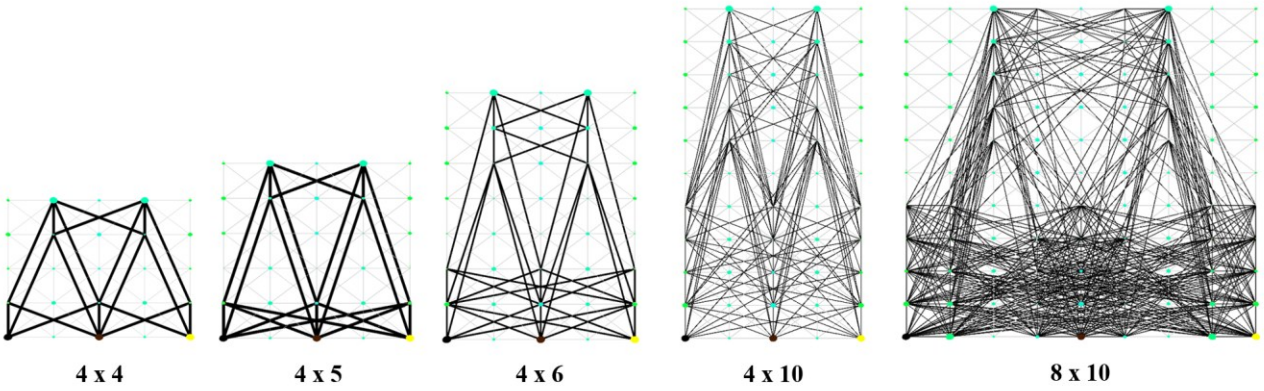

Figure 7. Scaling trusses without further evolution. The physical dimensions and substrate resolution (number of nodes in *x* and *y* axis) can be manually manipulated following evolution, and trusses retain many evolved characteristics.

Discussion

We present a novel extension of CPPN-NEAT for evolving functional 2-D network structures. We show that CPPN-NEAT with a dynamic substrate outperforms similar state-of-the-art HyperNEAT methods on four well-known benchmark problems for topology optimization, and more accurately approximates specific connectivity patterns of known global optima.

We acknowledge that none of the CPPN-based methods presented here are able to find the global optimum for these problems, unlike state-of-the-art evolutionary strategies, which *do* converge to global optima (as shown by Devert et al [20]). However, methods that rely on *direct* representations are prone to issues of (lack of) scalability, and are thus limited to addressing relatively simple design problems. *Generative* encodings will benefit various engineering domains [1-9] and ultimately allow us to exploit advanced manufacturing technologies and capabilities (such as high-value functionally graded solutions, multi-material composites, and so on.) In this context, "optimality" is less important than the ability to obtain *new types of morphology* that exhibit some specific functional behavior(s). CPPN-NEAT with dynamic substrate may hold important clues towards achieving this goal, and here we have presented an early proof-of-principle (albeit on a simple benchmark problem). In this concluding section, we describe additional properties of CPPN-NEAT with dynamic substrate that will offer useful trajectories for further research.

The key feature of our model is a more "biologically plausible" method of creating connectivity patterns. Instead of using HyperNEAT to define properties of specifically identified connections, we use a CPPN to control an additional network growth process. During this process, nodes use relative positional information to target surrounding nodes and grow locality biased connectivity patterns. In this paper, our growth step occurs just once to build 2-D truss designs. However, if we were to set our model within a physics-based simulation environment, nodes could continue to build and destroy connections over the lifetime of the structure, as nodes move. Such an approach could provide interesting *locomotive* behaviors that may be extremely robust to perturbation.

A second interesting implication of creating connections through growth is that we define new opportunities to exploit *node-based information*. For example, in the benchmark problems used in this paper, nodes are explicitly encoded with information that is used to (A) restrict degrees of freedom and (B) impose physical loads. In many engineering problems, this type of information is known well before optimization takes place, and could therefore easily be fed into CPPNs as additional information. Notably, Clune et al [24], have previously demonstrated that feeding additional information, relating to distance to center of a 3-D grid of voxels, can be exploited to create more rounded geometric features in solutions. Consequently, the ability to utilize information about boundary conditions may be useful in future work.

Thirdly, an exciting property of CPPNs is that the solutions they encode effectively obtain *infinite resolution*. For example, 2-D pictures evolved with CPPNs never get pixelated when they are scaled up. Similarly, truss designs evolved with CPPN-NEAT can be manipulated following evolution, and retain many of their characteristics (Fig 7). This property could have at least two useful applications. Firstly, the ability to increase resolution during evolution may help to optimize large-scale truss and/or network-based structures. Secondly, this added flexibility may provide new possibilities for *interactive* evolution, allowing network-based solutions to be explored and scaled in real-time. Indeed, in our current Java-based method, users can extract parameters of the underlying CPPN (connection weights) and manually adjust evolved solutions in real-time.

In addition to these avenues of enquiry, we will investigate the dynamical behavior of our model, in order to better understand (and improve) the method. In particular, we anticipate significant future savings in computational effort as we scale up the size of the substrate, due to our method avoiding non-essential connections between nodes. Future work will rigorously investigate the absolute performance improvement that may be obtained with our model on non-trivial problem instances.

Finally, further work is now underway to explore more complex 3-D problems with non-trivial mechanical properties (such as compliant mechanisms). Notably, the method presented in this paper can easily be extended to 3-D design by simply adjusting the FEM solver (see Fig 8).

References

[1] Dimas, L. S., Bratzel, G. H., Eylon, I., & Buehler, M. J. (2013). Tough composites inspired by mineralized natural materials: computation, 3D printing, and testing. *Advanced Functional Materials*, 23(36), 4629-4638.

[2] Cranford, S., & Buehler, M. J. (2010). Materiomics: biological protein materials, from nano to macro. *Nanotechnology, Science and Applications*, 3, 127.

[3] Lipson, H., & Kurman, M. (2013). *Fabricated: The New World of 3D Printing*. John Wiley & Sons.

[4] Fu, Q., Saiz, E., Rahaman, M. N., & Tomsia, A. P. (2011). Bioactive glass scaffolds for bone tissue engineering: state of the art and future perspectives. *Materials Science and Engineering: C*, 31(7), 1245-1256.

[5] Abdelaal, O. A., & Darwish, S. M. (2012). Analysis, Fabrication and a Biomedical Application of Auxetic Cellular Structures. *IJEIT*, 2(3), 218-223.

[6] Chu, C., Graf, G., & Rosen, D. W. (2008). Design for additive manufacturing of cellular structures. *Computer-Aided Design and Applications*, 5(5), 686-696.

[7] Oxman, N. (2011). Variable property rapid prototyping: Inspired by nature, where form is characterized by heterogeneous compositions, the paper presents a novel approach to layered manufacturing entitled variable property rapid prototyping. *Virtual and Physical Prototyping*, 6(1), 3-31.

[8] Menges, A. (2012). Material computation: Higher integration in morphogenetic design. *Architectural Design*, 82(2), 14-21.

[9] Hiller, J., & Lipson, H. (2012). Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, 28(2), 457-466.

[10] Bendsøe, M. P., & Kikuchi, N. (1988). Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*, 71(2), 197-224.

[11] Stanley, K. O., & Miikkulainen, R. (2003). A taxonomy for artificial embryogeny. *Artificial Life*, 9(2), 93-130.

[12] Rieffel, J., & Pollack, J. (2005). Automated assembly as situated development: using artificial ontogenies to evolve buildable 3-d objects. *Proceedings of GECCO 2005*, ACM, 99-106.

[13] Sigmund, O. (2011). On the usefulness of non-gradient approaches in topology optimization. *Structural and Multidisciplinary Optimization*, 43(5), 589-596.

[14] Shea, K., Aish, R., & Gourtovaia, M. (2005). Towards integrated performance-driven generative design tools. *Automation in Construction*, 14(2), 253-264.

[15] Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2), 131-162.

[16] Doursat, R., Sayama, H., & Michel, O. (2012). *Morphogenetic Engineering: Toward Programmable Complex Systems*. Springer.

[17] Kumar, S., & Bentley, P. J. (Eds.). (2003). *On Growth, Form and Computers*. Academic Press.

[18] Kicinger, R., Arciszewski, T., & De Jong, K. (2004). Morphogenesis and structural design: cellular automata representations of steel structures in tall buildings. *Proceedings of Evolutionary Computation, 2004. CEC2004*. Vol. 1, 411-418.

[19] Kowaliw, T., Grogono, P., & Kharma, N. (2007). The evolution of structural design through artificial embryogeny. *Proceedings of Artificial Life, 2007. IEEE Symposium on ALIFE'07*, 425-432.

[20] Devert, A., Weise, T., & Tang, K. (2012). A study on scalable representations for evolutionary optimization of ground structures. *Evolutionary computation*, 20(3), 453-472.

[21] Kicinger, R., Arciszewski, T., & Jong, K. D. (2005). Evolutionary computation and structural design: A survey of the state-of-the-art. *Computers and Structures*, 83(23), 1943-1978.

[22] Zavala, G. R., Nebro, A. J., Luna, F., & Coello, C. A. C. (2013). A survey of multi-objective metaheuristics applied to structural optimization. *Structural and Multidisciplinary Optimization*, 1-22.

[23] Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2), 99-127.

[24] Clune, J., & Lipson, H. (2011). Evolving 3d objects with a generative encoding inspired by developmental biology. *ACM SIGEVOlution*, 5(4), 2-12.

[25] Cheney, N., MacCurdy, R., Clune, J., & Lipson, H. (2013). Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. *Proceedings of GECCO 2013*, 167-174.

[26] Auerbach, J. E., & Bongard, J. C. (2012). On the relationship between environmental and mechanical complexity in evolved robots. *Artificial Life,* 13, 309-316.

[27] Szerlip, P., & Stanley, K. (2013). Indirectly Encoded Sodarace for Artificial Life. *Advances in Artificial Life, ECAL* Vol. 12, pp. 218-225.

[28] Stanley, K. O., D'Ambrosio, D. B., & Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2), 185-212.

[29] Verbancsics, P., & Stanley, K. O. (2011). Constraining connectivity to encourage modularity in HyperNEAT. *Proceedings of GECCO 2011*, pp. 1483-1490.

[30] Hornby, G. S., & Pollack, J. B. (2001). The advantages of generative grammatical encodings for physical design. *Proceedings of the 2001 Congress on Evolutionary Computation*, Vol. 1, pp. 600-607.

[31] Rieffel, J., & Smith, S. (2010). A Face-Encoding Grammar for the Generation of Tetrahedral-Mesh Soft Bodies. In *ALIFE,* 414-420.

[32] Fenton, M., McNally, C., Byrne, J., Hemberg, E., McDermott, J., & O'Neill, M. (2014). Automatic innovative truss design using grammatical evolution. *Automation in Construction*, 39, 59-69.

[33] Gauci, J., & Stanley, K. (2007). Generating large-scale neural networks through discovering geometric regularities. *Proceedings of GECCO 2009*, 997-1004.

[34] Richards, D., Dunn, N., & Amos, M. (2012). An evo-devo approach to architectural design. *Proceedings of GECCO 2012*, (pp. 569-576). ACM.

[35] Felippa, C. A. (2004). Introduction to Finite Element Methods. University of Colorado, Boulder, http://www.colorado.edu/engineering/CAS/courses.d/IFEM.d.

[36] Achtziger, W., & Stolpe, M. (2007). Truss topology optimization with discrete design variables—guaranteed global optimality and benchmark examples. *Structural and Multidisciplinary Optimization*, 34(1), 1-20.

[37] Bendsoe, M. P., & Sigmund, O. (2003). *Topology Optimization: Theory, Methods and Applications*. Springer.

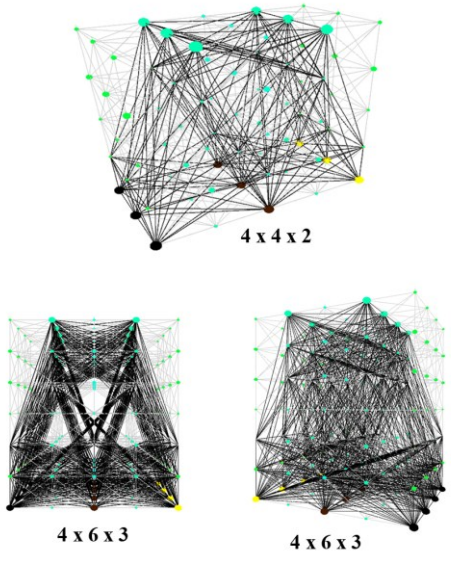

Figure 8. 2-D truss designs can easily be extended into 3-D.